SYSTEMS AND METHODS FOR A DATABASE ENGINE IN-PROCESS DATA PROVIDER

CROSS-REFERENCE(S) TO RELATED APPLICATION(S)

[0001] The present application is related by subject matter to the inventions disclosed in the following commonly assigned application: U.S. Patent Application No. (not yet assigned) (Atty. Docket No. MSFT-3030/307230.01), filed on even date herewith, entitled "SYSTEMS AND METHODS FOR A LARGE OBJECT INFRASTRUCTURE IN A DATABASE SYSTEM", the entirety of which is hereby incorporated herein by reference.

FIELD OF THE INVENTION

[0002] The present invention generally relates to the field of database systems and, more specifically, to systems and methods for enabling functions, procedures, and triggers for a database to be written in any of the .NET languages.

BACKGROUND OF THE INVENTION

[0003] Open Database Connectivity (ODBC) is an open standard application programming interface (API) for accessing a database. By using ODBC statements in a program, you can access files in a number of different databases, including Access,

dBase, DB2, Excel, and Text. In addition to the ODBC software, a separate module or driver is needed for each database to be accessed. The main proponent and supplier of ODBC programming support is Microsoft. ODBC is based on and closely aligned with The Open Group standard Structured Query Language (SQL) Call-Level Interface. It allows programs to use SQL requests that will access databases without having to know the proprietary interfaces to the databases. ODBC handles the SQL request and converts it into a request the individual database system understands.

[0004] Before an ODBC application can submit an SQL statement, it must allocate a statement handle for the statement. To allocate a statement handle, an application declares a variable of type HSTMT (for example, the application could use the declaration: HSTMT hstmt1) and then calls SQLAllocStmt to pass it the address of the variable and the connected HDBC with which to associate the statement. The driver allocates memory to store information about the statement, associates the statement handle with the hdbc, and returns the statement handle in the variable.

[0005] OLEDB is an application program interface (API) for access to different data sources. OLEDB includes not only the Structured Query Language (SQL) capabilities of the Microsoft-sponsored standard data interface Open Database Connectivity (ODBC) but also includes access to data other than SQL data. As a design from Microsoft's Component Object Model (COM), OLEDB is a set of methods (or routines) for reading and writing data. The objects in OLEDB consist mainly of a data source object, a session object, a command object, and a rowset object. An application using OLEDB would use this request sequence: (i) initialize OLE; (ii) connect to a data source; (iii) issue a command; (iv) process the results; and (v) release the data source

object and uninitialize OLE. (While OLE once stood for "Object Link Embedding" and "DB" for database, many skilled artisans no longer ascribes these meanings to the letters "OLE" and "DB.")

[0006] In OLEDB, ICommand contains methods to execute commands. A command can be executed many times, and the parameter values can vary. This interface is mandatory on commands. A command object contains a single text command, which is specified through ICommandText, an interface that is mandatory on commands, and a command object can have only one text command. When the command text is specified through ICommandText::SetCommandText, it replaces the existing command text.

[0007] ActiveX Data Object (ADO) is an application program interface from Microsoft that lets a programmer writing Windows applications get access to a relational or non-relational database from both Microsoft and other database providers. For example, to write a program that would provide users of a Web site with data from an IBM DB2 database or an Oracle database, one could include ADO program statements in an HTML file that are then identified as an Active Server Page; then, when a user requested the page from the Web site, the page sent back would include appropriate data from a database, obtained using ADO code.

[0008] OLEDB is the underlying system service that a programmer using ADO is actually using. A feature of ADO, Remote Data Service, supports "data-aware" ActiveX controls in Web pages and efficient client-side caches. As part of ActiveX, ADO is also part of Microsoft's overall Component Object Model (COM), its component-oriented framework for putting programs together. ADO instruction objects are termed commands as the usual intention is for these commands to be performed immediately.

The most elemental responsibility of a command object is to perform the operations intended by an instruction. Additional responsibilities might include: undoing operations; performing the operations in a different context, possibly in a different process, machine, or time schedule (batching); pr simulating the effect of the operations. For programming languages that do not provide threading, running batched CommandObjects can provide pseudo-threading.

[0009] ADO.net is a set of classes in .NET that allows for data access. ADO.net is comprised of classes found in the System.Data namespace that encapsulate data access for distributed applications. However, rather than simply mapping the existing ADO object model to .NET to provide a managed interface to OLEDB and SQL Server, ADO.net changes the way data is stored and marshaled within and between applications. The primary reason ADO.net redefines this architecture is that most applications developed today can benefit from the scalability and flexibility of being able to distribute data across the Internet in a disconnected fashion.

[0010] Because the classic ADO model was developed primarily with continuously connected access in mind, creating distributed applications with it is somewhat limiting. A typical example is the need to move data through a Recordset object between tiers in a distributed application. To accomplish this in classic ADO you have to specifically create a disconnected Recordset using a combination of properties including cursor location, cursor type, and lock type. In addition, because the Recordset is represented in a proprietary binary format, you have to rely on COM marshalling code built into OLEDB to allow the Recordset to be passed by value (ByVal) to another component or client code. This architecture also runs into problems when attempting to

pass recordsets through firewalls because these system level requests are often denied. On the other hand, if you elected not to use disconnected recordsets, you had to devise your own scheme to represent the data using Variant arrays, delimited within a string, or saved as tabular XML using the Save method (although the latter option is really only viable when using ADO 2.5 and higher). Obviously these approaches have their downside because they run into problems with performance and maintainability not to mention interoperability between platforms. In addition, the classic ADO model doesn't handle hierarchical data particularly well. Although it is possible to create hierarchical recordsets using the Microsoft data shape provider, it is not simple and is therefore not often used. Typically JOIN clauses are used inside stored procedures or inline SQL to retrieve data from multiple tables. However, this does not allow you to assemble data from multiple data sources and easily determine from where the data comes. As a result, classic ADO provides a flat view of data that is not strongly typed. To alleviate these problems, ADO.net is built from the ground up for distributed applications used in today's disconnected scenarios. For example, the central class in ADO.net is the DataSet, which can be thought of as an in-memory XML database that stores related tables, relationships, and constraints. As you'll see, the DataSet is the primary mechanism used in VB.NET applications to cache data and pass it between tiers in a distributed application thereby alleviating the need to rely on proprietary schemes or COM marshalling.

[0011] Using XML alleviates several of the burdens of classic ADO. For example, by storing the data as XML it can easily pass through firewalls without special configuration. In addition, by storing related tables and representing the relationships

between those tables the DataSet can store data hierarchically allowing for the easy manipulation of parent/child relationships. The self-describing nature of XML combined with the object-oriented nature of VB.NET also allows for direct programmatic access to the data in a DataSet in a strongly typed fashion. In other words, the data need not be accessed using a tables, rows, and columns metaphor but can be accessed in terms of the definition of the data that can be type checked by the compiler. Furthermore, this disconnected model combined with connection pooling schemes frees resources on the database server more quickly, allowing applications to scale by not holding on to expensive database connections and locks.

[0012] Web services (sometimes called application services) are services that are made available from a business's Web server for Web users or other Web-connected programs. Providers of Web services are generally known as application service providers. Web services range from such major services as storage management and customer relationship management (CRM) down to much more limited services such as the furnishing of a stock quote and the checking of bids for an auction item. The accelerating creation and availability of these services is a major Web trend. Users can access some Web services through a peer-to-peer arrangement rather than by going to a central server. Some services can communicate with other services and this exchange of procedures and data is generally enabled by a class of software known as middleware. Services previously possible only with the older standardized service known as Electronic Data Interchange (EDI) increasingly are likely to become Web services. Besides the standardization and wide availability to users and businesses of the Internet itself, Web services are also increasingly enabled by the use of the Extensible Markup

Language (XML) as a means of standardizing data formats and exchanging data. XML is the foundation for the Web Services Description Language (WSDL).

[0013] .NET is a collection of programming support for web services, the ability to use remote services rather than your own computer for various services. The purpose of .NET is to provide individual and business users with a seamlessly interoperable interface for applications and computing devices and to make computing activities increasingly browser-oriented. The .NET platform includes servers; building-block services, such as web-based data storage; and device software.

[0014] The .NET Common Language Runtime (CLR) is an execution platform that manages the execution of intermediate language (IL) code generated from any one of several programming languages, and the CLR allows these different IL code components to share common object-oriented classes written in any of the supported languages—for example, the CLR allows an instance of a class written in one language to call a method of a class written in another language. Moreover, programs compiled for the CLR do not need a language-specific execution environment, and these programs can easily be moved and executed on any system with CLR support.

[0015] Programmers developing code in Visual Basic, Visual C++, or C# compile their programs into intermediate language code called Common Intermediate Language (CIL) in a portable execution (PE) file that can then be managed and executed by the CLR. The programmer and the environment specify descriptive information about the program when it is compiled and the information is stored with the compiled program as metadata. This metadata, stored in the CIL compiled program, tells the CLR what

language was used, its version, and what class libraries will be needed by the program for execution.

[0016] The CLR also provides services, such as automatic memory management, garbage collecting (returning unneeded memory to the computer), exception handling, and debugging, and thus provides a powerful yet easy-to-use programming model. The CLR is sometimes referred to as a "managed execution environment" (MEE), and the IL code that executes within the CLR is called managed code. Currently a CLR running on a server that also hosts a DBMS is problematic because the CLR can compromise the reliability, scalability, security, and robustness of the DBMS. For example, when operating independently on the same server, both the DBMS and CLR manage memory, threads, and synchronization between multiple threads, and sometimes conflicts can result. Various embodiments of the present invention provides a solution to this problem.

[0017] In computer programming, a connection is the setting up of resources (such as computer memory and buffers) so that a particular object such as a database or file can be read or written to. Typically, a programmer encodes an OPEN or similar request to the operating system that ensures that system resources such as memory are set up, encodes READs and WRITES or similar requests, and then encodes a CLOSE when a connection is no longer needed so that the resources are returned to the system for other users.

[0018] Previous releases of SQL Server provided extensibility through extended stored procedures. To access data from the local instance developers had to use a data

access API (ODBC/OLEDB) to loop back to the server and perform data access being agnostic to the condition of running inproc as part of an already established connection.

[0019] In a database, procedures and functions performs a distinct service. The language statement that requests the function is called a function call. Programming languages usually come with a compiler and a set of "canned" functions that a programmer can specify by writing language statements. These provided functions are sometimes referred to as library routines. Some functions are self-sufficient and can return results to the requesting program without help. Other functions need to make requests of the operating system in order to perform their work.

[0020] In a database, a trigger is a set of Structured Query Language (SQL) statements that automatically "fires off" an action when a specific operation, such as changing data in a table, occurs. A trigger consists of an event (an INSERT, DELETE, or UPDATE statement issued against an associated table) and an action (the related procedure). Triggers are used to preserve data integrity by checking on or changing data in a consistent manner.

[0021] T-SQL (Transact-SQL) is a set of programming extensions that add several features to the Structured Query Language (SQL) including transaction control, exception and error handling, row processing, and declared variables. Microsoft's SQL Server and Sybase's SQL server support T-SQL statements.

[0022] Heretofore, to run application code in a relational database management system (RDBMS), such code had to be Transact-SQL (TSQL) code. The alternative was for an application run its code in its current location and call-up the data it was

processing, row by row, from the corresponding database. Neither approach is efficient, and what is needed is an alternative approach that is both efficient and effective.

SUMMARY OF THE INVENTION

[0023] Various embodiments of the present invention enable functions, procedures, and triggers to be written in any of the .NET languages and executed by the RDBMS. User code can access data from the local or other SQL servers using the SQL Programming Model and both the SqlServer or SqlClient implementations respectively.

[0024] To improve upon the previous extensibility mechanism, a set of APIs (commonly known as "the in-process provider" or "inproc provider") was developed to provide efficient and easy to use data access while running inproc. As a whole, the inproc provider is a very easy to use data access API, an implementation of the ADO.net programming model. The present invention utilizes this API in this regard.

BRIEF DESCRIPTION OF THE DRAWINGS

[0025] The foregoing summary, as well as the following detailed description of preferred embodiments, is better understood when read in conjunction with the appended drawings. For the purpose of illustrating the invention, there is shown in the drawings exemplary constructions of the invention; however, the invention is not limited to the specific methods and instrumentalities disclosed. In the drawings:

[0026] Fig. 1 is a block diagram representing a computer system in which aspects of the present invention may be incorporated;

[0027] Fig. 2 is schematic diagram representing a network in which aspects of the present invention may be incorporated;

[0028] Fig. 3 is a block diagram illustrating the brute-force approach for converting code into TSQL statements for execution in a RDBMS.

[0029] Fig. 4 is a block diagram illustrating the row-by-row approach to copying the necessary data up to the application for local processing.

[0030] Fig. 5 is a block diagram illustrating the method of various embodiments of the present invention to transmit .NET code from an application to a RDBMS for direct execution and return of results to said application.

DETAILED DESCRIPTION OF ILLUSTRATIVE EMBODIMENTS

[0031] The inventive subject matter is described with specificity to meet statutory requirements. However, the description itself is not intended to limit the scope of this patent. Rather, the inventor has contemplated that the claimed subject matter might also be embodied in other ways, to include different steps or combinations of steps similar to the ones described in this document, in conjunction with other present or future technologies. Moreover, although the term "step" may be used herein to connote different elements of methods employed, the term should not be interpreted as implying any particular order among or between various steps herein disclosed unless and except when the order of individual steps is explicitly described.

Computer Environment

[0032] Numerous embodiments of the present invention may execute on a computer. Fig. 1 and the following discussion is intended to provide a brief general description of a suitable computing environment in which the invention may be implemented. Although not required, the invention will be described in the general

context of computer executable instructions, such as program modules, being executed by a computer, such as a client workstation or a server. Generally, program modules include routines, programs, objects, components, data structures and the like that perform particular tasks or implement particular abstract data types. Moreover, those skilled in the art will appreciate that the invention may be practiced with other computer system configurations, including hand held devices, multi processor systems, microprocessor based or programmable consumer electronics, network PCs, minicomputers, mainframe computers and the like. The invention may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote memory storage devices.

[0033] As shown in Fig. 1, an exemplary general purpose computing system includes a conventional personal computer 20 or the like, including a processing unit 21, a system memory 22, and a system bus 23 that couples various system components including the system memory to the processing unit 21. The system bus 23 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. The system memory includes read only memory (ROM) 24 and random access memory (RAM) 25. A basic input/output system 26 (BIOS), containing the basic routines that help to transfer information between elements within the personal computer 20, such as during start up, is stored in ROM 24. The personal computer 20 may further include a hard disk drive 27 for reading from and writing to a hard disk, not shown, a magnetic disk drive 28 for reading from or writing to a removable magnetic disk 29, and an optical disk drive 30 for reading

from or writing to a removable optical disk 31 such as a CD ROM or other optical media. The hard disk drive 27, magnetic disk drive 28, and optical disk drive 30 are connected to the system bus 23 by a hard disk drive interface 32, a magnetic disk drive interface 33, and an optical drive interface 34, respectively. The drives and their associated computer readable media provide non volatile storage of computer readable instructions, data structures, program modules and other data for the personal computer 20. Although the exemplary environment described herein employs a hard disk, a removable magnetic disk 29 and a removable optical disk 31, it should be appreciated by those skilled in the art that other types of computer readable media which can store data that is accessible by a computer, such as magnetic cassettes, flash memory cards, digital video disks, Bernoulli cartridges, random access memories (RAMs), read only memories (ROMs) and the like may also be used in the exemplary operating environment.

[0034] A number of program modules may be stored on the hard disk, magnetic disk 29, optical disk 31, ROM 24 or RAM 25, including an operating system 35, one or more application programs 36, other program modules 37 and program data 38. A user may enter commands and information into the personal computer 20 through input devices such as a keyboard 40 and pointing device 42. Other input devices (not shown) may include a microphone, joystick, game pad, satellite disk, scanner or the like. These and other input devices are often connected to the processing unit 21 through a serial port interface 46 that is coupled to the system bus, but may be connected by other interfaces, such as a parallel port, game port or universal serial bus (USB). A monitor 47 or other type of display device is also connected to the system bus 23 via an interface, such as a video adapter 48. In addition to the monitor 47, personal computers typically include

other peripheral output devices (not shown), such as speakers and printers. The exemplary system of Fig. 1 also includes a host adapter 55, Small Computer System Interface (SCSI) bus 56, and an external storage device 62 connected to the SCSI bus 56.

[0035] The personal computer 20 may operate in a networked environment using logical connections to one or more remote computers, such as a remote computer 49. The remote computer 49 may be another personal computer, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to the personal computer 20, although only a memory storage device 50 has been illustrated in Fig. 1. The logical connections depicted in Fig. 1 include a local area network (LAN) 51 and a wide area network (WAN) 52. Such networking environments are commonplace in offices, enterprise wide computer networks, intranets and the Internet.

[0036] When used in a LAN networking environment, the personal computer 20 is connected to the LAN 51 through a network interface or adapter 53. When used in a WAN networking environment, the personal computer 20 typically includes a modem 54 or other means for establishing communications over the wide area network 52, such as the Internet. The modem 54, which may be internal or external, is connected to the system bus 23 via the serial port interface 46. In a networked environment, program modules depicted relative to the personal computer 20, or portions thereof, may be stored in the remote memory storage device. It will be appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers may be used. Moreover, while it is envisioned that numerous embodiments of

the present invention are particularly well-suited for computerized systems, nothing in this document is intended to limit the invention to such embodiments.

Networking Environment

[0037] Fig. 2 illustrates an exemplary network environment in which the present invention may be employed. Of course, actual network and database environments can be arranged in a variety of configurations; however, the exemplary environment shown here provides a framework for understanding the type of environment in which the present invention operates.

[0038] The network may include client computers 20a, a server computer 20b, data source computers 20c, and databases 70, 72a, and 72b. The client computers 20a and the data source computers 20c are in electronic communication with the server computer 20b via communications network 80, e.g., an Intranet. Client computers 20a and data source computers 20c are connected to the communications network by way of communications interfaces 82. Communications interfaces 82 can be any one of the well-known communications interfaces such as Ethernet connections, modem connections, and so on.

[0039] Server computer 20b provides management of database 70 by way of database server system software, described more fully below. As such, server 20b acts as a storehouse of data from a variety of data sources and provides that data to a variety of data consumers.

[0040] In the example of Fig. 2, data sources are provided by data source computers 20c. Data source computers 20c communicate data to server computer 20b via

communications network 80, which may be a LAN, WAN, Intranet, Internet, or the like. Data source computers 20c store data locally in databases 72a, 72b, which may be relational database servers, excel spreadsheets, files, or the like. For example, database 72a shows data stored in tables 150, 152, and 154. The data provided by data sources 20c is combined and stored in a large database such as a data warehouse maintained by server 20b.

[0041] Client computers 20a that desire to use the data stored by server computer 20b can access the database 70 via communications network 80. Client computers 20a request the data by way of SQL queries (e.g., update, insert, and delete) on the data stored in database 70.

In-Process Data Provider

[0042] T-SQL (Transact-SQL) is a set of programming extensions that add several features to the Structured Query Language (SQL) including transaction control, exception and error handling, row processing, and declared variables. Microsoft's SQL Server and Sybase's SQL server support T-SQL statements.

[0043] Heretofore, to run application code in a relational database management system (RDBMS), such code had to be Transact-SQL (TSQL) code. Fig. 3, which illustrates the brute-force approach for this method, shows an application 302 running on a local operating system 304 on local hardware 306 that, via a network 310, connects with a RDBMS 312 running on a server operating system 314 on server hardware 316 and having a database 318 in, for example, a persistent data store 320. For the application 302 to run code 308 on the RDBMS 312, said code 308 must be converted

into TSQL code 308' by some code-converting method/mechanism 322 that may include manual coverting, automated converting, or the like; however, conversion is time-consuming, prone to errors, and provides no mechanism (denoted by 324) for easy introduction of the converted TSQL code 308' into the RDBMS 312.

[0044] One alternative has for an application to run its code in its current location and call-up the data it was processing, row by row, from the corresponding database. This method is illustrated in the block diagram of Fig. 4 where an application 302 calls 402 for a first data row 432 via the network 310 from the RDBMS 312, and the RDBMS transmits 404 the first row (not shown) from the database 318 to the application 302 via the network 310 for the application 302 to process. The application then repeats this process for each subsequent row until the entirety of data needed is requested downstream and returned upstream in the system to replicate the data at the applications location 406—a very inefficient and ineffective approach.

[0045] Various embodiments of the present invention are directed to enabling functions, procedures, and triggers written in any of the .NET languages to be executed directly in the RDBMS. User code can access data from the local or other SQL servers using the SQL Programming Model and both the SqlServer or SqlClient implementations respectively. As illustrated in Fig. 5, the .NET application 302' submits 502 its code through the network 310 directly to the RDBMS 312—the functionality of which is enhanced with the ADO.NET in-process provider 555—for execution on the database 318 and the result (if any), when completed, is transmitted 504 from the RDBMS 312 via the network 310 to the .NET application 302'.

[0046] In this way, functions, procedures, and triggers can be written in any of the .Net languages, and this invention provides for a programming model that affords: (1) the acknowledgment of the invocation context and how it was exposed and made available to end users ; and (2) the separation of immutable and mutable parts of a request in order to be able to achieve higher efficiencies.

Invocation Context

[0047] For several embodiments of the present invention, managed code may be invoked in the database server whenever a user calls a procedure, calls a function, or whenever a user's action fires a trigger defined in any of the .NET languages. Since execution of this code is requested as part of user connection, it is necessary to have access to the caller's context from the code running in the server as certain data access operations may only be valid if run under the caller's context (e.g., access to inserted and deleted pseudo-tables or lock sharing). In this regard, the caller's context is consolidated and exposed in a single class, SqlContext, that provides access to the following components:

- Connection: For a user to send commands to the server that user must first establish a connection, and thus any code running in the server has an associated caller's connection context.
- Command: A command sent to the server has a context of execution with state, such as SET options, database context, execution variables, and so forth. This context is exposed in different ways through the APIs, that is, the context

is mapped to a statement handle (hstmt) in ODBC, an ICommand and/or ICommandText in OLEDB, and command objects in ADO and ADO.net.

- Transaction: Every statement in the server runs in the context of a transaction. It could be either running under an explicitly started transaction, an implicitly started transaction, or a single statement auto transaction.
- Pipe: The "pipe" through which requests and results flow to and from the client and server is part of the context. It is made available through the programming model in order for users to be able to send information back.
- TriggerContext: Can only be retrieved from a CLR trigger, and provides information about the operation that caused the trigger to fire and a map of the columns that were updated.

Static Requests

[0048] For general efficiency, the SQL Server execution model attempts to cache and reuse as much as possible. For stored procedures and functions written in CLR languages, the type of requests executed are generally constant except for a few parameter values that vary by execution. In addition, ithe ADO.net programming model there is no clear separation between execution invariants (that can be cached and reused) and per-execution parameters—that is commands, parameter metadata, and parameter data are all inter-mingled.

[0049] For several embodiments of the present invention, changes have been made to the programming model to (a) allow users to construct requests in the way they are familiar with; (b) explicitly identify a request as being "defined and ready for reuse";

and (c) specify at runtime only the information required. To this end, a new type of object, SqlDefinition, is used to represent a "compiled" command, and this object should be thought of as a SqlCommand that is immutable and which holds no runtime state. As such, a SqlDefinition object can be reused across multiple executions. Moreover, to simplify the learning curve for user, certain embodiments allow SqlDefinitions to be created by simply passing a SqlCommand to an a constructor.

[0050] With the static part of a request being abstracted into SqlDefinition's, a new CreateCommand() overload was added to the SqlConnection object for various embodiments of the present invention, and this overload returns a SqlCommand that is ready to be used by only setting the appropriate input parameter values.

[0051] An example of one model for these various embodiments is as follows:

```
public static SqlDefinition PrepareRequest()
{
     SqlCommand cmd = new SqlCommand();
     cmd.CommandType=CommandType.Text;

cmd.CommandText = "INSERT products values (@ProdName);
SET @ProdId=SCOPE_IDENTITY()";

          cmd.Parameters.Add(new
SqlParameter("@ProdName"));
          cmd.Parameters[0].SqlDbType=SqlDbType.NVarchar;
cmd.Parameters[0].Size=200;

cmd.Parameters.Add(new SqlParameter("@ProdId"));
```

```
    cmd.Parameters[1].Direction=ParameterDirection.Output;
        cmd.Parameters[1].SqlDbType=SqlDbType.Int;

return new SqlDefinition(cmd);
}

public static ReadOnly SqlDefinition mySP =
PrepareRequest();

public static SqlInt32 InsertProduct(SqlString Description,
ref SqlInt32 ProdId)
{
    //Get a SqlCommand given the definition stored in the
above reference
    Using (SqlCommand sp=
SqlContext.GetConnection().CreateCommand(mySP))

{
        //Set the value for the input parameter
        sp.SetSqlString(0, Description);

        //Execute the command
        sp.ExecuteNonQuery(SqlContext.GetTransaction());

        //Get the output parameter
        ProdId=sp.GetSqlInt32(1);
}
}
```

[0052] The division between static and dynamic parts of a request allows better caching and reuse of metadata and associated unmanaged structures. Also the model gives a declarative sense to the procedural .Net programming languages, bringing the overall programming model closer to TSQL.

Cursor on "Everything"

[0053] The ADO.net API has a DataReader API which exposes a forward-only cursor on top of SQL statement execution results, and this API also exposes a Read() method that advances the cursor to the next row. However, in between Read() calls, user code needs to be allowed to run, and this API in and of itself does not provide for this ability.

[0054] For several embodiments of the present invention, and in order to provide an opportunity for user code to be run between Read() calls, all TSQL statements are divided into two groups, "single-steppable" (SS) and "non-single-steppable" (NSS). For SS statements, a row is fetched and buffered (as a shallow copy) and control is returned to the caller unwinding the stack while preserving all state; upon request for the next row, execution state is restored and the next row is processed; and so forth.

[0055] For NSS statements, a two-thread pipe model is used such that a second thread is spawned to execute the statement and synchronization is performed between the two threads such that rows that are produced by the second thread (the "sub-thread") are made available one row at a time to the first thread (the "user-code-thread").

[0056] A special handoff of the transaction context is done in the two-thread pipe model to allow both threads to run under the same transaction. No intra-transaction synchronization is needed though since it is known that, at any given time, at most one of the two threads will be executing. Therefore, based on the steppability of the statements, each frame decides at compile time the execution strategy to be used, and the design switches between the two execution modes giving full cursor support on all statements yet is completely transparent to the caller.

[0057] Lastly, when an NSS statement is in a frame, the whole batch is deemed to be non-steppable and the two-thread pipe model is employed for the entire frame.

Out-of-Process Provider Symmetry

[0058] For the in-process provider of the present invention to fully enables the ADO.net programming model in the database, certain functionality must be supported by the API. Therefore, the in-process API is fully symmetry with the full API as implemented by an out-of-process provider (SqlClient). Thus, for several embodiments of the present invention, one or more of the following features native to the out-of-process provider (SqlClient) are enabled:

- MARS: The in-process provider would support more than one pending executing command per connection. This entails the support to have multiple active stacks within a single connection. Conceptually this would enable a tree of stacks in the server within a single client side request. For this to be enabled, the top level server side frame would be assumed to have the "default" execution context, which would be cloned for each starting sub-

request. Upon completion of the sub-requests, execution environment would be copied back to the "default" context, exposing semantics consistent to those exposed with top level client-side MARS. Also similar to how it is done for client-side MARS, the in-process provider's multiple stacks will share the transaction context with other substacks.

- Autonomous Transactions: The API exposes the standalone concept of a transaction that can be freely associated to one or more requests that are to be executed. Based on the infrastructure provided by the unified transaction framework, the in-process provider would expose multiple top level transactions that can be associated with multiple commands at a given time.
- Cancel / Attention: The API exposes the ability to cancel an executing request which maps to the ability to unwind one of the possible substacks and return to the next higher CLR frame.
- Debugging: Hooks and debugger stops are included with the in-process provider to notify an attached debugger of the transition between TSQL and CLR frames. In this way, the in-process provider supports mixed debugging that allows end users to seamlessly step between managed code and TSQL.

Data Marshaling

[0059] For several embodiments of the present invention, efficient data marshaling is one of the key aspects to the performance achieved by the in-process provider design. In this design, small scalar values are copied from unmanaged to managed space by avoiding all intermediate unnecessary copies. However, the

components design for marshaling of large types (MAX data types, XML, and large user-defined types) entaild additional built-in capabilities as follows:

- Mutable managed types SqlBytes and SqlChars are created to be able to maximize instance reuse. Mutability enables instances to be reused reducing need for allocations and hence for garbage collection, latency and CPU use.
- These types can be backed by unmanaged pointers and/or interfaces. As such, they can be constructed and handed off to the user, and are backed by the unmanaged ILockBytes without need to marshal the entire value into managed space.
- For instances that are writable, SqlBytes and/or SqlChars implement a copy-on-write mechanism such that only the unamanged reference is wrapped but, upon an intent to change the underlying value, a copy is triggered and the mutation is performed entirely on managed space.
- A similar copy-on-write mechanism is implemented for parameter passing of large values.

[0060] In general the in-process provider keeps track of unmanaged data being referenced from managed space, and prevents access of this data outside of the intended managed frame.

Conclusion

[0061] The various systems, methods, and techniques described herein may be implemented with hardware or software or, where appropriate, with a combination of both. Thus, the methods and apparatus of the present invention, or certain aspects or

portions thereof, may take the form of program code (i.e., instructions) embodied in tangible media, such as floppy diskettes, CD-ROMs, hard drives, or any other machine-readable storage medium, wherein, when the program code is loaded into and executed by a machine, such as a computer, the machine becomes an apparatus for practicing the invention. In the case of program code execution on programmable computers, the computer will generally include a processor, a storage medium readable by the processor (including volatile and non-volatile memory and/or storage elements), at least one input device, and at least one output device. One or more programs are preferably implemented in a high level procedural or object oriented programming language to communicate with a computer system. However, the program(s) can be implemented in assembly or machine language, if desired. In any case, the language may be a compiled or interpreted language, and combined with hardware implementations.

[0062] The methods and apparatus of the present invention may also be embodied in the form of program code that is transmitted over some transmission medium, such as over electrical wiring or cabling, through fiber optics, or via any other form of transmission, wherein, when the program code is received and loaded into and executed by a machine, such as an EPROM, a gate array, a programmable logic device (PLD), a client computer, a video recorder or the like, the machine becomes an apparatus for practicing the invention. When implemented on a general-purpose processor, the program code combines with the processor to provide a unique apparatus that operates to perform the indexing functionality of the present invention.

[0063] While the present invention has been described in connection with the preferred embodiments of the various figures, it is to be understood that other similar

embodiments may be used or modifications and additions may be made to the described embodiment for performing the same function of the present invention without deviating there from. For example, while exemplary embodiments of the invention are described in the context of digital devices emulating the functionality of personal computers, one skilled in the art will recognize that the present invention is not limited to such digital devices, as described in the present application may apply to any number of existing or emerging computing devices or environments, such as a gaming console, handheld computer, portable computer, etc. whether wired or wireless, and may be applied to any number of such computing devices connected via a communications network, and interacting across the network. Furthermore, it should be emphasized that a variety of computer platforms, including handheld device operating systems and other application specific hardware/software interface systems, are herein contemplated, especially as the number of wireless networked devices continues to proliferate. Therefore, the present invention should not be limited to any single embodiment, but rather construed in breadth and scope in accordance with the appended claims.

[0064] Finally, the disclosed embodiments described herein may be adapted for use in other processor architectures, computer-based systems, or system virtualizations, and such embodiments are expressly anticipated by the disclosures made herein and, thus, the present invention should not be limited to specific embodiments described herein but instead construed most broadly.